〔非公開〕

ＴＲ－Ｃ－００５６

プロダクションシステムとＡＴＭＳによる画像認識システムＡＳＤＳ

栄藤　稔　　岸野　文郎
MINORU ETOU　FUMIO KISHINO

１９９０．８．２９

ＡＴＲ通信システム研究所

プロダクションシステムと ATMS による
画像認識システム ASDS

栄藤 稔　岸野 文郎
Minoru ETOH and Fumio KISHINO
ATR 通信システム研究所 知能処理研究室
ATR Communication Systems Research Laboratories

1990 年 8 月 28 日

要 旨

当研究所で開発された画像認識システム ASDS(Assumption-based Scene Description System) の実現手法，プログラミングについて報告する．ASDS は室内シーン理解を行なうことを目標としている．特に人間身体等の一般化円筒の部分クラスで近似できるような物体の認識を知識処理の観点から行なうことを特徴としている． ASDS では一般化円筒部分クラスと直方体を構成する平面をシーンの 3 次元記述要素として得た後，モデルとの照合によりシーン解釈を行なう．その処理内容は RGB カラー画像の処理から，モデルとの照合まで及ぶ． 画像処理の構造化にともなうヒューリスティクスをルールベース化し， ATMS によりヒューリスティクスの衝突による矛盾を多重世界で解消している．その実現に際しては，複数のワークステーション上で動作する物理的構成上に市販のエキスパートシェル ART， SUN Common-LISP, C を用いた．

目次

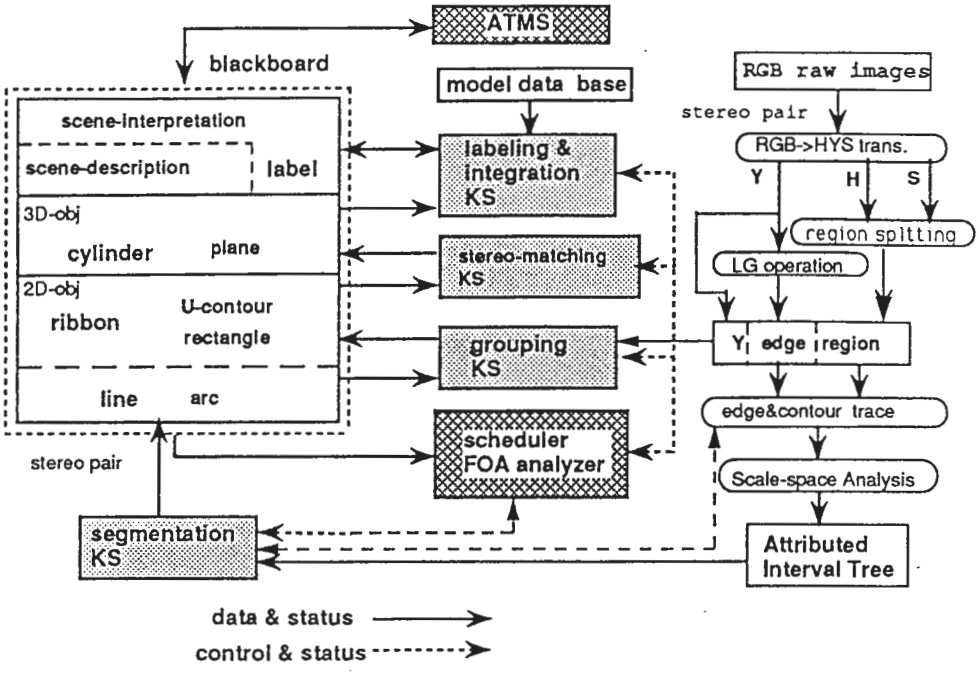

図 1: ASDS のアーキテクチャ

1 システムの全体構成

図 1 に開発された実験システム (Assumption-based Scene Description System, ASDS)[5] の論理的な構成を示す.

このデータフローは次のようなステップからなっている.

1. 領域抽出 :

   RGB のステレオ画像から, 色相 H, 輝度 Y, 彩度 S に変換し, まず彩度を用いて閾値処理により領域を 2 分する. さらに彩度の高い領域を色相のヒストグラム解析により分割する. これは本研究所の宮脇 [1] らの研究成果をそのまま用いている.

2. エッジ抽出 :

   領域抽出のあと, 明度画像 Y からは Marr-Hildreth のラプラシアンオペレータ [22] を用いてエッジを求める. このときゼロ交差点についてコントラストを求めておき, コントラストがある閾値以上のものを選び, その点から連結すべき画素を探索する. 閾値以下の画素は後に必要に応じて探索される. コントラストの値は Sobel フィルターの出力で得ている.

3. チェインコードの生成 :

   以上の領域外縁, エッジの尾根上を 3 × 3 のマスクを操作してチェインコードを生成する.

4. スケールスペース解析 :

   得られたチェインコードをスケールスペース解析して, 木構造にする. この木構造は AIT(Attributed Interval Tree) と呼ぶ.

5. 画像の基本特徴の生成 :

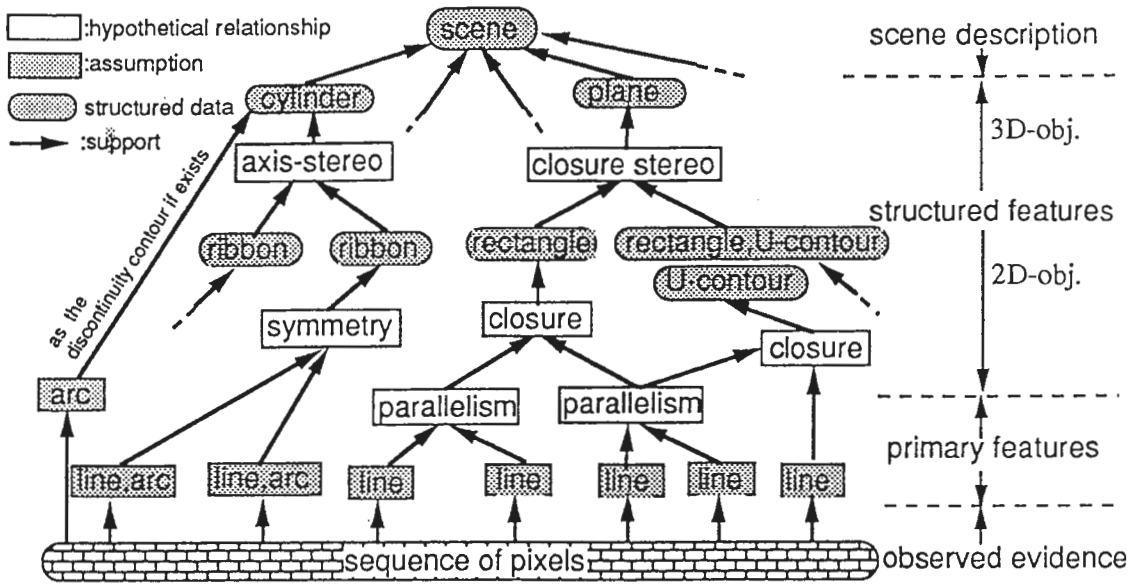

図 2: シーン記述の階層

AIT 中から 線分と円弧データを 直線度，円弧度を評価して 所定の閾値を満たすものを生成する． 当初はこれをルールで記述していたが，現在は LISP 関数で実現している．

6-1. 画像の基本特徴の構造化 ：

このステップより，ルールベースに書かれた処理が始まる．基本特徴の空間関係により， 基本特徴を統合して 3 次元のシーン記述要素（シリンダと平面）を生成する．空間関係として“平行”，“対称”，“近接”，“共線”，“類似”の関係を用いる． これらをルールとして記述してある．

6-2. モデルとの照合 ：

モデルの部分データと得られた 3 次元のシーン記述要素とを照合する． これはステップ 6-1 と同時に行なわれる．

図 2 に ASDS で扱われるデータの階層を示す．シーンの記述とは構造化特徴 (structured feature) の矛盾のない集合を意味する． また円弧，線分を総称して基本特徴 (primary feature) と呼ぶ．シーン記述の階層の中で，シリンダーと平面は仮定された基本特徴間でさらに 空間関係を仮定することに よって生成される． 図中，矩形で囲まれた項目は仮説であり，楕円で囲まれた項目はグループ化とステレオ照合により得られた記述要素である．

図 3 に ASDS の物理的構成を示す．ローカルマシンは エキスパートシェル ART が動作する UNIX マシンならどれでも良い．スケールスペースの計算サーバーとして複数のリモートマシンを用いている． さらに認識結果の表示用として シリコングラフィックスの IRIS を用いている．

2 サブシステム

前章で示した 処理ステップの中で，領域分割，エッジ抽出，スケールスペース解析，信頼度， グルーピング，ステレオ，モデルとの照合の各々の技術的内容を項目別に説明する．

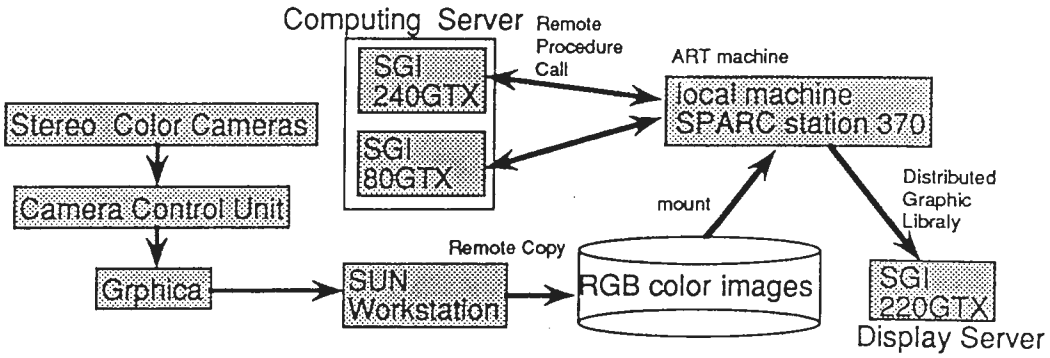

図 3: ASDS の物理的構成

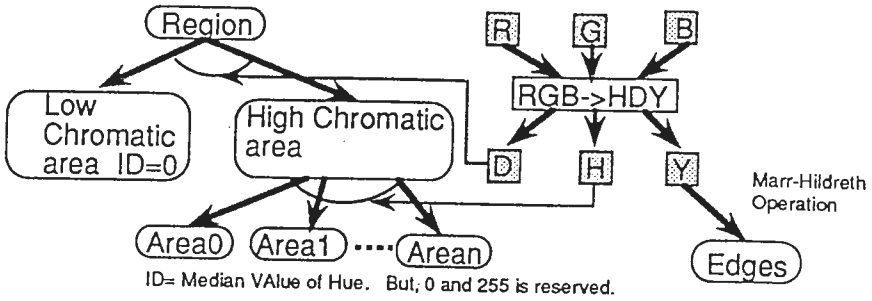

図 4: 領域分割の処理の流れ

2.1 領域分割

処理の技術内容は宮脇らの文献 [1] を参考にされたい．領域分割の概略を図 4 に示す． 宮脇らの手法では RGB 画像から HVD なる表色系を用いて領域分割を試みる．これは以下のステップからなっている．

1. RGB 画像を彼らの提案する HVD 表色系へ変換する．
   H は色相， V は画素 (x,y) について $V(x,y) = max(R(x,y), G(x,y), B(x,y))$ として得られ， D は

   $$D(x,y) = max(R(x,y), G(x,y), B(x,y)) - min(R(x,y), G(x,y), B(x,y))$$

   として得られる． 通常 彩度にはこの D を V の値で正規化したものを用いるが，低輝度画素の彩度が不安定なため，正規化を行なっていない．

2. D の値により領域を 2 分割する．この閾値を thr_D とする． thr_D 未満の領域は V の値により領域を分割を行なう． thr_D 以上の領域は次ステップ以降で色相により分割処理をおこなう．

3. thr_D 以上の領域について，横軸に色相を縦軸に画素数をとりヒストグラムを作成する． ヒストグラムについて，色相の近傍について，画素数の加算平均をとることにより平滑化を行なう． 近傍は $h_{smooth} = 2n + 1$,n は 1 より大きい自然数となる．

4. ヒストグラムの中で，ピークを見つけ，ピーク値が閾値 thr_P 以上であれば，ピークの前後画素を領域とする．

5. ピーク解析から得られた画素に色相値を ID を与え，領域とする．孤立点がある場合は 8 近傍画素の値に値を変更する．

A	B	C
D	E	F
G	H	I

図 5: Sobel オペレータに用いる近傍画素

ASDS では色相のみによる領域分割を行なうことから， RGB 画像から V は用いず，色相 H，彩度 D の画像のみを生成する．
なお，パラメータは thr_D, h_{smooth} thr_P の 3 つである．

2.2 エッジ抽出

エッジ抽出は 2 通りの方法が用意されている． Mar-Hildreth のエッジオペレータと Sobel オペレータである． RGB 画像から

$$Y(x,y) = (77R(x,y) + 151G(x,y) + 28B(x,y))/256 \tag{1}$$

の式により輝度画像を作り これに上記のオペレータを作用させる．

2.2.1 Sobel Operator

図 5 において，

$$D_x = (C + 2F + I) - (A + 2D + G) \tag{2}$$

$$D_y = (A + 2B + C) - (G + 2H + I) \tag{3}$$

$$|D| = \sqrt{{D_x}^2 + {D_y}^2} \tag{4}$$

として $D(x,y)$ を得る． エッジは $|D(x,y)|$ のピーク（尾根）として得ることができる．

2.2.2 Marr-Hildreth Operator

2 次元のガウス分布は

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\pi\sigma}) \tag{5}$$

として表される．ここで $r^2 = x^2 + y^2$ とすると ガウスフィルタのラプラシアンは

$$\nabla^2 G(r) = \frac{r^{-2\sigma^2}}{2\pi\sigma^2}\exp(-\frac{r^2}{2\sigma^2}) \tag{6}$$

と表現される． σ をパラメータとしてフィルタ係数を算出し，画像とのコンボルーションを行なう． このとき零交差点を求めることによってエッジの抽出を行なう． ASDS では Marr-Hildreth オペレータの零交差によりエッジを求める． 但し，零交差は明度変化の不連続な極大点全てに生じるため，多くの不必要なエッジが得られる． このためにコントラストを導入する． 零交差点について前述の Sobel オペレータを作用させ，その出力絶対値をコントラスト値とする．コントラスト値を閾値処理することによって必要なエッジを得ることができる．これは画像特徴抽出の標準技術である．

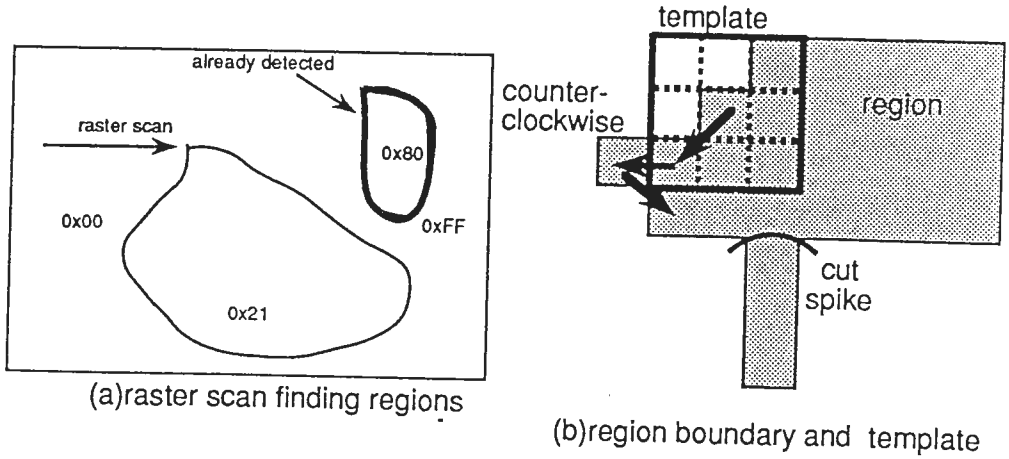

図 6: 領域外縁の探索

2.2.3 スケールスペース解析

得られた領域，エッジを線分及び円弧データに変換するために （輪郭）線曲率のスケールスペース解析を用いる．これは文献 [2] に詳しい．ここでは補足事項のみを述べる．

最初に 得られた領域の外縁，エッジを探索してチェインコードとする．このチェインコードは 3 項組になっていて，C 言語の構造体で表せば

```
struct  IN_DATA{
        int     x;      /* X座標値 */
        int     y;      /* Y座標値 */
        int     t;      /* 全曲率 (deg) */
};
```

となる．全曲率はその点における接線方向の傾きであり，0° ～ 360° にの範囲である． 接線方向の基準は探索開始時の接線方向とし，それからの相対的な方向を全曲率としている．最終目的が，輪郭曲率の変曲点を求めることから，全曲率の代わりに 「k 曲率 (k-curvature)」 [7] を求めても良い．位置 l の全曲率を $\tau(l)$，k 曲率を $\kappa(l)$ とすると輪郭変曲点は $\tau(l) * \nabla^2 G(\sigma, l)$ あるいは $\kappa(l) * \nabla G(\sigma, l)$ のゼロ交差点として得ることができる．後者の場合，k 曲率を $k = 1$ としてゼロ交差を得る時，そのゼロ交差位置は前者とほぼ同じである． 現在は全曲率を求めてから，2 階微分により変曲点を求めている． この点で，文献 [2] とは異なる．

エッジと領域外縁の探索に関して補足を加える．

外縁の符号化 輪郭外縁は図 6 に示すように 8bit の深さを持つ入力画像のコピーにおいて，左上から領域 ID(1 から 254 まで) をラスター走査により探すことによって始まる． 0 は低彩度領域として予約されている．また 255 は既に探索された領域のマスクとして使用する． 0 と 255 以外の領域外縁を反時計周りに 8 連結の方向で探索してチェインコードを生成する． コードの長さ 3 以下は出力しない． また (b) に示すような連結画素が高々 2 しかない外縁 (spike) は，マスクを端点からバックトレイスすることによって検出し， 除去する．

エッジの符号化 エッジの符号化はコントラスト値の影響を受けることで外縁探索と異なる．コントラストに対して

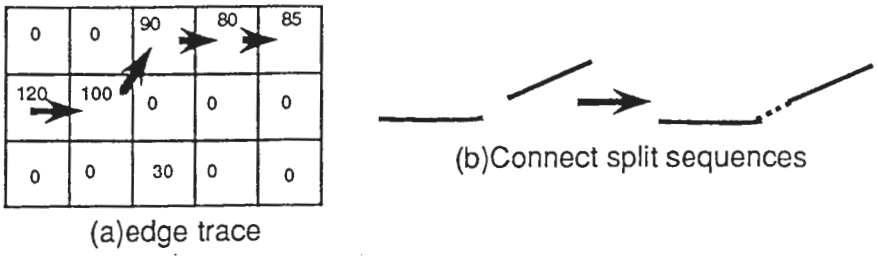

図 7: エッジの探索

閾値 th_{seed} と th_{trace} を設ける．エッジ画像のコピーのなかで，まず th_{seed} 以上の値を持つ画素が 3 画素以上連結している位置を探索する．この画素の中点から th_{trace} 以上の画素が 8 近傍にあれば必ずコードに連結する（図 7 左）．8 近傍に分岐がある場合は現在のコントラスト値に最も近い画素を連結する． なお エッジは途切れている場合が多い． 従って，最初に得られたチェインコード（フラグメントと呼ぶ）を以下のヒューリスティクスで連結する（図 7 右）．

- 端点間の距離が 2 画素以内で傾きの内積が 0.6 以上
- 端点間の距離が 10 画素以内で傾きの内積が 0.75 以上

以上のようにして得られたチェインコードをスケールスペース解析することにより，曲率極点の階層構造 Curvature Extremal Points Structure(CEPS) を作る． CEPS はノードの集合 $\mathbf{V}$ とノード間リンクの集合 $\mathbf{E}$ の 2 項組で表現される．ノードは各スケールでの輪郭曲率の極大, 極小点に相当し， $(l, \sigma, type, \kappa_\sigma)$ なる 4 項組で表現される．

- l : 輪郭線位置（輪郭始点位置からの画素数）
- σ: スケール
- $type$: $\{m^+, M^+, m^-, M^-\}$ の 4 つのタイプを持つ．また $\{M^+, M^-\}$ を極大 M, $\{m^+, m^-\}$ を極小 m と略すときがある．
- κ_σ: そのスケールのノードにおける曲率

$\mathbf{V}$ はこのノード集合であり, 異なるスケール尺度 σ により $\sigma 0$ を最下位として昇順に階層化されている．

$$\mathbf{V} = \bigcup_{i=0}^{N-1} \mathbf{V}_{\sigma i}$$

連続するスケールのノード集合間 $\mathbf{V}_{\sigma i}, \mathbf{V}_{\sigma i+1}$ にはリンクがある。

$$\mathbf{E} = \{(v_0, v_1) | v_0 \in \mathbf{V}_{\sigma i}, v_1 \in \mathbf{V}_{\sigma i+1}, 0 \leq i \leq N-2\}$$

CEPS は以下の処理ステップで生成される．

1. ノードの生成: 輪郭線上位置 l の接線方向 $\tau(l)$ の系列に対してガウシアンフィルター $G(\sigma, l)$ とその 1 階微分 $\nabla G(\sigma, l)$ と 2 階微分 $\nabla^2 G(\sigma, l)$ のコンボルーションをとり, σ について変化させながらスケールスペースに展開する．2 次微分波形 $\kappa(l) * \nabla^2 G(\sigma, l)$ のゼロ交差点をノードとし, 属性を与える． この時ノードの輪郭線位置は暫定的にそのスケールにおけるゼロ交差位置とする．最終的な輪郭線位置は次のリンク処理が終了した時点で決定され

る．またゼロ交差位置を曲率一定の区間においても安定して得るために，適当な幅 $\pm\kappa'_\delta$ を 1 次微分波形が通過する区間の中点として得ている．

2. 特徴点のリンク: スケールスペース上の特徴点はスケーリングを大きくするにつれて平滑化によって極大，極小値はその絶対値を小さくしながら，対になって消失していく．隣接する極大ノード，極小ノードを M, m，消失を示す記号，ϕ で表すと

$$mM \Longrightarrow \phi$$

の消失規則となる．その他の特徴点は同じ極大，極小のタイプを保持しながら上位（σ が大なる方向）へ引き継がれる．即ちスケールが前後するノード v_0, v_1 間で $type(v_0) = type(v_1), type(v) = \{M, m\}$ となるようにリンクをはる．このリンクは一意に得ることはできない．そこで次のようなサブステップにより行なった．

(a) $\mathbf{V}_{\sigma i}$ に属するすべてのノードをその曲率の絶対値によりソートする．

(b) 曲率絶対値の大きいものから，同じタイプである特徴点を隣接する $V_{\sigma j}, (\sigma i < \sigma j)$ の同じ輪郭位置から $\pm n\sigma$ の範囲で探索する．この範囲はスケールの刻みにもよるが経験的に $n = 2$ 程度で行なっている．

(c) もし所定の範囲になければ，その特徴点は消失である．

(d) 消失と判定された特徴点全てが消失規則を満足していればこのスケールでのリンク処理は終了する．

このサブステップを最下位の $\sigma 0$ から繰り返し，最上位まで行なってリンクの処理が終了する．最後に最下位のノードの輪郭線位置をリンクのある上位のノードへ伝搬させていくことによって各ノードの暫定輪郭線位置を更新する．

2.3 信頼度

ASDS で仮説として扱われるデータには信頼度に相当する重み付けがある．例えば 2 つの線分の対称性を評価するために，線分間の距離，傾き等をパラメータとしてその信頼度を求める．ここで異なるパラメータから得られた信頼度を Dempster-Shafer の確率論 (以下 D-S と略す) により融合する．

D-S により，仮説が信じられている，信じられていないという 2 つの状態を考えると、$\{IN, OUT\}$ なる普遍集合が規定できる。そこで D-S 理論に基づき Belief B(IN) と Disbelief DB(IN) を考える．ここで

$$B(IN) = lower\ probability(IN)$$

$$DB(IN) = 1 - upper\ probablity(IN)$$

$$DB(IN) = B(OUT)$$

である．この belief と disbelief の 2 項組を D-S 値と便宜的に呼ぶことにする．普遍集合内の排反命題は D-S 理論の背中律によって信頼度が計算される．また異なる証拠からの D-S 値の更新では D-S 理論により次のように計算される．

仮説 a について異なる環境で得られた D-S 値 (B1,DB1) と (B2,DB2) の融合は以下の式となる (Ginsberg[6] も同様の計算手法を示している)．

$$B(a) = \frac{B1*(1-DB2)+(1-DB1)*B2-B1*B2}{1-(B1*DB2+DB1*B2)} = 1 - \frac{1-(1-B1)*(1-B2)}{1-(B1*DB2+B2*DB1)}$$

$$DB(a) = \frac{DB1*(1-B2)+(1-B1)*DB2-DB1*DB2}{1-(B1*DB2+DB1*B2)} = 1 - \frac{1-(1-DB1)*(1-DB2)}{1-(B1*DB2+B2*DB1)}$$

これを COMB 結合と呼ぶ．

例えばリボンの信頼度は 方向性，対称性，近接性の 3 点から得られた信頼度をこの COMB 結合から得ている．

```
(direction (fuzzy-z-function-to-DS
    (abs (sin (angle-lines line0 line1))) -0.2 0.4 1.0))
(symmetry (fuzzy-s-function-to-DS
   (abs (/ (line-length axis)
   (+ (line-length line0) (line-length line1))))
   0.2 0.35 0.5))
(proximity (if (< (max (line-length base0) (line-length base1)) 10)
     (return-from symmetry? '(0 1))
     (fuzzy-z-function-to-DS
      (/ (+ (line-length base0) (line-length base1))
 (line-length axis)) 0 6 12)))
(belief (comb-belief (list symmetry proximity direction))))
```

COMB 結合の基となる値には “fuzzy-z-function-to-DS” や “fuzzy-s-function-to-DS” 等の ヒューリスエティクスの関数を用いて行なっている． 例えば向き合う 2 線分の平行性を評価したい場合を考える． 2 線分のなす角度の正弦値が 1 であれば 2 線分は平行である． 逆に 0 であれば垂直である．ここで，平行であれば確率 1 でこの 2 線分はリボンを形成するという保証はない． また，平行から垂直にいたる中間値の設定も必要である．そこで，図 8 に示すように ファジィ推論で用いられる S 関数， Z 関数を用い，これを D-S の (belief,disbelif) の組に割り当てる． z 関数の a,b,c の設定を -0.2,0.4,1.0 とすると，垂直で D-S 値は (0,1) となり，他の対称性や，近接性の D-S 値を COMB 結合しても (0,1) となる，一方，平行の場合を考えると D-S 値は (0.88888896 0) となり，他の尺度との COMB 結合により総合的に判定される．以下に z 関数と， s 関数の定義を LISP で示す．

```
;;; ファジィ理論における Z-関数
;;;u が a 以下で 1， b で 0.5， c 以上で 0 s-function の逆
(defmacro fuzzy-z-function (u a b c)
  `(- 1 (fuzzy-s-function ,u ,a ,b ,c)))

;;; ファジィ理論における S-関数
;;;u が a 以下で 0， b で 0.5， c 以上で 1
(defun fuzzy-s-function (u a b c)
  (if (or (< b a) (< c b)) (error "~A~%" "fuzzy-s-function arguments error"))
  (cond ( (<= u a) 0)
        ( (<= u b)
          (let ((tmp (/ (- u a) (- c a))))
            (* 2 tmp tmp)))
        ( (<= u c)
          (let ((tmp (/ (- c u) (- c a))))
            (- 1 (* 2 tmp tmp))))
        ( t 1)))
```

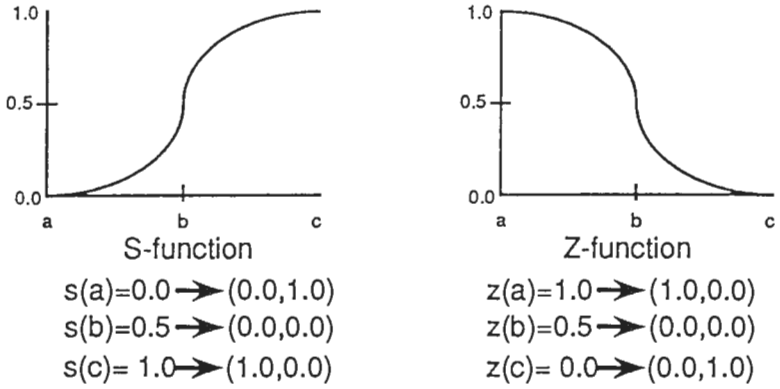

図 8: ファジィ S 関数と Z 関数からの D-S 値への割り当て

2.4 グルーピング

ASDS では得られた線分と円弧（基本特徴）をその平行性 (parallelism)，近接性 (proximity)，共線 (colllinearity) 性等の位置関係により，画像上で構造化する．図 9 に構造化処理を示す．これらの信頼度は 前節の信頼度計算法によって行なわれる．

2.5 ステレオ

シーンの 3 次元記述要素として，一般化円筒の軸と多面体を構成する平面を求めることになる． このための手段としてステレオ法を用いる．図 10 および図 11 に円筒軸と平面の 3 次元位置回復の原理を示す．一般化円筒の軸が求まる条件として，中心軸の投影が，円筒表面が形成する輪郭 (extremal contour) の対称軸に一致しなければならないという制約がある． ここで一般化円筒の中で，中心軸について回転体，あるいは直線の中心軸に対して断面形状一定で点対称という部分クラスがこの制約を満足する [4]．この対称性の制約は厳しく見えるが，実際，人物や花瓶等は部分的にシリンダーで粗く近似でき，本手法でも人の手や姿勢の 3 次元認識に適用可能である [2][4]． 3 次元平面 (plane) については,3 次元平面の抽出は閉領域を囲む辺を単に平面の境界であると仮定して，閉領域を構成する辺のうち，より垂直な (90° ±45° 以内) 2 辺のステレオ対応をとり，これによって得られた空間中の 2 線分によって平面を決定する（図 11）． より垂直な 2 辺を用いたのはステレオ照合による誤差がエピポーラ線に垂直であるほど少ないことによる．

3 ART と ATMS

ART はルールベースとそれを駆動する推論エンジン，さらにデータをフレーム構造（ART ではスキーマと呼ぶ）で管理することが可能なエキスパートシェルである． ART はさらに複数の排反する可能世界を管理するビューポイントと呼ばれる機構を持っているが，推論速度（時間あたりのルールの発火回数）の低下が予測されたこと，システム構築の柔軟性が損なわれること 等の理由により， ATMS を LISP プログラムとして作成し，フレームデーモンとして組み込んだ． ATMS は複数コンテクストでの命題の信念状態を管理する機構であり，推論結果の無矛盾性を保証し，その成立する仮定の範囲を同時に保持する． ATMS の詳しい動作の説明はその原著 [16] に譲るとして，ここでは ATMS の用語説明と意味付けを行なった後，ルールの動作を説明する． ATMS は導出ルールと仮説生成ルールの帰結に対して以下の 3 項組からなる 「ノード」を生成する．

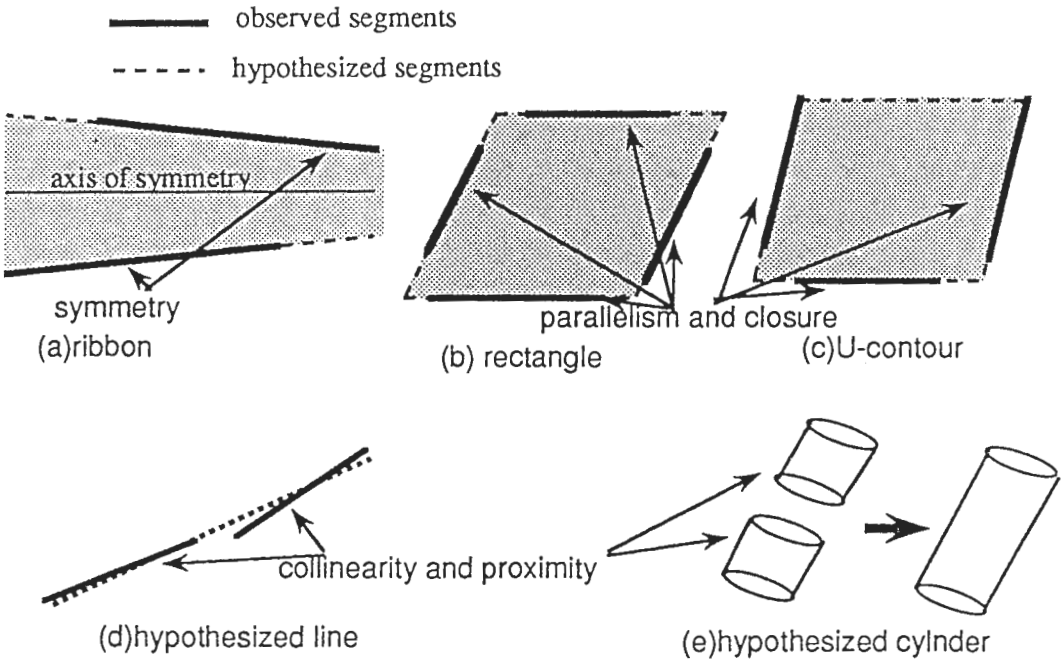

図 9: グルーピング

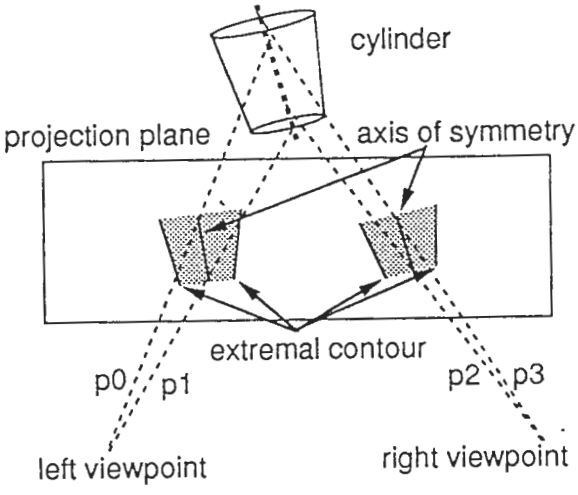

図 10: 軸対応によるシリンダーの回復

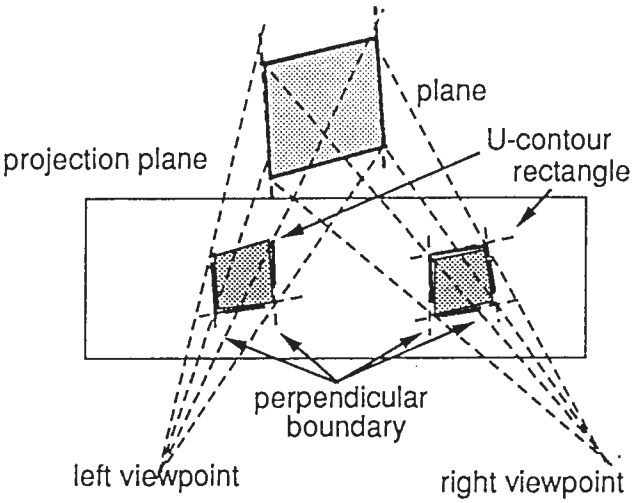

図 11: 2 辺対応による平面の回復

$<$ データ, ラベル, 正当化 $>$.

「データ」はルールの発火により得られた帰結の内容を示している. 一般的に「ラベル」は 「環境」の選言的集合からなっているが, ここでは構造化特徴が前提部の連言から導出されていることから, ラベルは単一の環境から構成されている. 環境は矛盾のない仮説の連言的集合からなっている. これによってデータの依存関係と成立する範囲が表現される. 「正当化」はそのデータを導出するに至った前提と帰結の組を表している. ASDS ではラベルが特別な空集合 $\{\phi\}$ からなる環境を持つ時, データは常に恒真である「事実」として扱われる. 仮定はそれ自身を正当化するデータであり, そのラベル中の環境は仮定のデータそのものを要素とする. ある環境 (矛盾のない仮説の集合) から導かれる全ての帰結を 「コンテクスト (context)」と呼ぶ. 本論文では特に, コンテクストの中で, 極大即ち, 他のいかなる帰結を含めても矛盾となるコンテクストを「極大コンテクスト」と呼ぶことにする.

ATMS はデータの信念状態を返す. 信念状態は そのデータがいずれかのコンテクストに属しているとき "IN" であると表現し, 現在信じられていることを意味する. またデータがいずれのコンテクストにも属していないとき 信念状態は "OUT" であると表現し, もはや信じられていないことを意味する. ATMS では矛盾のある仮説の組合せを "NOGOOD" として, テーブルとして保持している. ATMS は新たな環境が生成された時, その環境が NOGOOD テーブル内の仮説集合の上位集合となることを検査することによって信念状態を判定する. ASDS では シーン記述・解釈の各要素と ATMS との対応を表 1 に示すようにとる.

一方, ルールは次のように条件部が各条件の連言からなる形式をしている.

条件 1, 条件 2, ..., 条件 n $\Rightarrow$ $<$ 実行 $>$

ルールの 3 つの形式, 導出ルール, 仮説生成ルール, 制約ルールはその実行部に 3 つの異なる手続き Justify, Hypothesize と Nogood を持つことが異なる. 以下に各形式におけるルールと ATMS との相互作用を述べる. なおデータベースとは図 1 中の "blackboard" を指す. またルール中の変数項 X, Y 等はルールの発火により命題として扱われる.

- 導出ルール (justificatory rule)

ルール形式 : $X_0, X_1, ..., X_n \Rightarrow$ Justify : Y

もし Y が 前提 $X_0, X_1, ..., X_n$, から帰結として導かれる時はJustify 手続きによって Y がデータベースに宣言される.

ATMS : ATMS は新たなノードを次のように生成する.

$$< Y, \{\{\bigcup_i^n env(X_i)\}\}, \{\{X_0, X_1, ..., X_n\}\} > .$$

関数 $env(X)$ は X のラベルに含まれる環境を返す.

データベース : もし環境 $\bigcup_i^n env(X_i)$ がNOGOOD テーブルに含まれるいかなる環境の上位集合でないなら, Y はINである.

- 仮説生成ルール (hypothesizing rule)

ルール形式 : < 観測 > $\Rightarrow$ Hypothesize : X

データベース中の観測に基づいて具体化された X を仮定する.

ATMS : ATMS は新たなノードを以下に生成する.

$$< X, \{\{X\}\}, \{\{X\}\} > .$$

データベース : X は後に制約ルールによりそれ自身がNOGOOD であると宣言されない限りINである.

以上の変形として ラベル内の環境を特別な空集合 $\{\phi\}$ とする

$$< X, \{\{\phi\}\}, \{\{X\}\} > .$$

なるノードを生成する手続き Assert によって, 観測から事実を言明することができる.

- 制約ルール (constraint rule)

ルール形式 : < X が矛盾 > $\Rightarrow$ Nogood: X

ATMS : ATMS は NOGOOD テーブルに環境 $env(X)$ を加える. そしてテーブルの内容を環境の極小性を保ちながら更新する.

これにより, ATMS の真理管理機能により環境 $env(X)$ を含む全ての環境は矛盾となる.

データベース : X と $env(X)$ の上位集合をレベルにもつデータの全てが OUT となる.

なお, プログラムソースコードでは Justify, Hypothesize, Assert, Nogood は各々 justify!, assume!, asssert!, nogood! となっている (開発の歴史的経緯による). ASDS ではルールの実行に際して各ルールは条件部の各データがIN 状態であることを検査する. その意味で, Forbus の言う「IN 戦略」[21] に相当する. また導出ルールでは発火に先だって条件部によって束縛されたデータのラベル内の環境の積が矛盾でないかどうかを検査する. これによって, 信念状態が当初から OUT である無駄なデータの生成を避けている. ルール発火やコンテクストの選択はスケジューラが行ない, ATMS は信念状態の管理のみを行なう.

図1のブラックボードはデータベース内の論理的な構成であり, 実際は ART 上の単一のデータベース内に格納される. ART の機能としてデータをフレーム構造で表現できることから, データ間の is-a 関係, instance-of 関係が容

易に実現できる．また，スロット値の変更に伴うデーモンを設定することができる（フレームデーモンの概念はこれより広く，スロットの生成，消滅等にも関与できる．正確にはこの機能を “active-value” と ART では呼んでいる）．全てのデータは “atms-data” の下位クラスとして次のように定義されている．

```
(defschema atms-data
  (status IN)
  (atms-node)
  (belief))

(defschema atms-daemon
  (is-a active-value))

(add-active-value 'atms-data 'atms-node 'atms-daemon)

(defaction put-after (atms-daemon)
  (atms-data atms-node node)
  (unless node (error "atms-daemon put-after can't work!"))
  (let ((label (tms-node-label node))
(status (get-schema-value atms-data 'status)))
    (if (label-contradictory? label)
      (and (eq status 'IN) (referent-schema-value atms-data 'status 'OUT))
      (and (eq status 'OUT) (referent-schema-value atms-data 'status 'IN)))))
```

ここでは “atms-daemon” を定義している．これは宣言されたデータの信念状態を ATMS に問い合わせて，信念状態に変化があれば “status” スロットを書き換える動作を行なう．次に線分データの設定を次に示す．線分データは atms-data の is-a であることが示されている． 従って先ほどの atms-daemon はこのデータのインスタンスについて全て動作する． さらにここでは表示用のデーモン “display-cylinder” を定義している．これにより，線分データの信念状態により描画が制御される．

```
(defschema line-data
  "画像データの基本となる直線データのスキーマ"
  (is-a atms-data)
  (head)        ;start
  (tail)        ;end
  (length)    ;length oh the line
  (orientation) ;
  (plane)   ;other information which a user defines
  (type)
  (hue)
  (interval)
  (pixels)
```

```
  (prior 0)
  )

(defschema display-line
  (is-a active-value))
(add-active-value 'line-data 'atms-node 'display-line)

(defaction (put-after after) (display-line)
  (line-data atms-node node)
  (when (and (slotp line-data 'head) (slotp line-data 'tail))
        (let
            ((start-point (mapcar #'round
                                  (list*$ (get-schema-value line-data 'head))))
             (end-point   (mapcar #'round
                                  (list*$ (get-schema-value line-data 'tail))))
             (side   (get-schema-value line-data 'plane))
             (matrix (evaluate-transformation-matrix
                      0 0 ;origin
                      0 0 ;translation
                      0   ;rotation
                      1 1 ;(x,y) scale factors
                      nil))
             (mode (if (in? line-data) *draw* *erase*)))
          (when (and start-point end-point)
                (line (first start-point) (second start-point)
                      (first end-point) (second end-point)
                      1
                      mode
                      (if (eq side 'LEFT)
                          'left-contour-plane 'right-contour-plane)
                      matrix
                              t) ))))
```

ATMS と ART の関係を図 12 に示す.

このように図 2 に示したデータは各クラスのインスタンスとしてデータベース内に投入される. 線分データと対称関係の実例を図 13 に示す. このようにスキーマ名として一意に発生された名前をとり, INSTANCE-OF スロットにより, 各データのクラスの実例であることが判る.

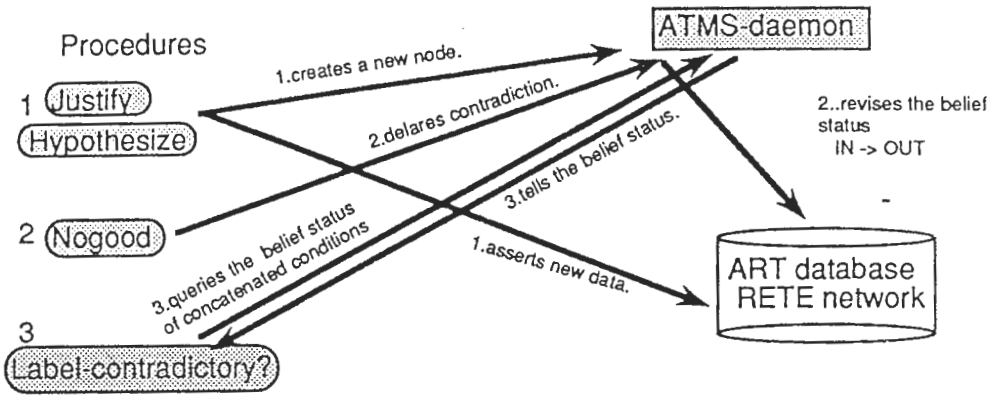

図 12: ATMS, ART, ルール

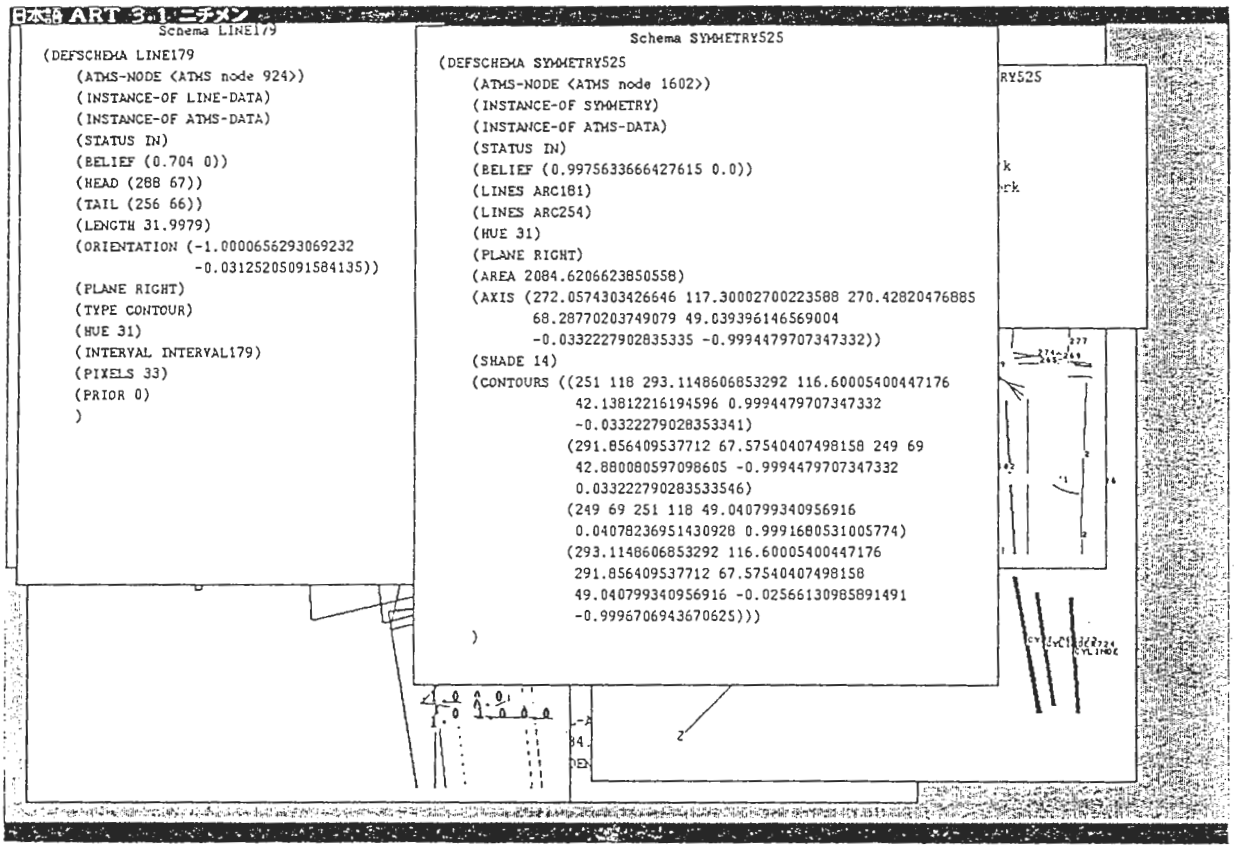

図 13: ART データベース実例

4 Low-level subsystem

画像の低レベル処理に関する処理モジュールを説明する． ここで低レベル処理とは 1 章の領域抽出，エッジ抽出の 2 つの処理を言う．これらの プログラムは $src/preprocess/splitting に格納されている．ファイルは必要な C プログラムは

```
data_mgr.c@   incsort.c     lg_filter.c   ope3x3.c      peaklow.c
dummy_main.c  io.c@         miyahist.c    opedre.c      pre_filter.c
getthr.c      lastimg.c     noizu.c       outlog.c      splitting.c
```

であり．これを Call する LISP プログラムは

```
pre_filter.lisp  sample.lisp
```

である． sample.lisp はその名のとうり， SUN Common-LISP の環境から領域抽出，エッジ抽出を行なうことができるようになっている． pre_filter.lisp に LISP からのインタフェース関数が示されている．

最重要なインタフェース関数を以下に示す．

```
(define-c-function (pre-filtering ''_pre_filter'')
  ((red-data-filename   :string)   ; R 画像データファイル名
   (green-data-filename :string)   ; G 画像データファイル名
   (blue-data-filename  :string)   ; B 画像データファイル名
   (smooth              :integer)  ; smoothing parameter
   (minthr              :integer)  ; min.threshold
   (dth                 :integer)  ; 色彩情報に作用するしきい値
   (sgm                 :integer)  ; LG フィルタリングで使う σ
   (lg-thr              :integer)  ; LG フィルタの SOBEL 値用しきい値
   (mode                :integer)) ; LG フィルタリングの処理モード
  :result-type :pointer)
```

リターン値は C 関数の構造体に対する ポインタである．

```
/* 前処理結果を返すためのデータ構造      */
struct  PRE_FILTER_DATA{
  unsigned char *y_data;  /* 輝度情報テーブルへのポインタ */
  unsigned char *edge;    /* 輪郭情報テーブルへのポインタ */
  unsigned char *region;  /* 領域情報テーブルへのポインタ */
};
```

パラメータは領域分割の技術的説明で述べた thr_D, h_{smooth} thr_P の 3 つと，エッジ抽出に関して sgm,lg-thr,mode の 3 つがある． sgm は ガウシアンフィルターの係数を決定するものである．

- mode =0 であれば エッジ上で 255，それ以外で 0 の画像が結果となる

- mode =1 であれば エッジ上でコントラスト値，それ以外で 0 の画像が結果となる
- mode=2 であれば LG オペレーションの途中結果が結果となる.

エッジ上の判定は ゼロ交差位置でコントラスト値が lg-thr 以上であればエッジ画素となる．以上のパラメータの標準的値は

```
(setq red-data-filename   ''../data/hito_1.r'')
;; R 画像データファイル名
(setq green-data-filename ''../data/hito_1.g'')
;; G 画像データファイル名
(setq blue-data-filename  ''../data/hito_1.b'')
;; B 画像データファイル名
(setq smooth 5)      ;; smoothing parameter
(setq minthr 52)     ;; min.threshold
(setq dth    25)     ;; 色彩情報に作用するしきい値
(setq sgm    300)    ;; σとして指定したい数値を１００倍した整数を与える
(setq lg-thr 10)     ;; LG フィルタの SOBEL 値用しきい値
(setq mode   1)      ;; LG の処理モード
```

である.

5 Mid-level subsystem

中レベル処理とはチェインコード生成とスケールスペース解析の 2 つの処理であり，画像データを記号化する．これに関係する処理は $src/preprocess/ の下に格納されている.

```
$src/preprocess/ の下
arc_fitting.c                    line_detect.c
config.h                         line_detect.h
damy_lisp.c                      make_ai_graph_data.c
damy_lisp.h                      make_out_line_graph_data.c
data_mgr.c                       matching.c
data_mgr.h                       matching.h
define.h     others.c
dummy_edge.c                     psoc.c
edge_detect.c                    psoc.h
extern.h                         semapho_mgr.c
fitting.h                        syoukai.c
fitting1.c                       tcp_tansaku.c
global.h                         tcp_tansaku.h
init_end.c                       test.c
```

```
io.c
co_psoc/co_psoc.c                co_tansaku/co_tansaku.c
co_psoc/config.h@                co_tansaku/config.h@
co_psoc/data_mgr.c@              co_tansaku/data_mgr.c@
co_psoc/data_mgr.h@              co_tansaku/data_mgr.h@
co_psoc/define.h@                co_tansaku/define.h@
co_psoc/extern.h@                co_tansaku/extern.h@
co_psoc/rpc_tcp.h                co_tansaku/semapho_mgr.c@
co_psoc/rpc_xdr.c                co_tansaku/tcp_tansaku.h@
co_psoc/tcp_tansaku.h@
for_ciris/config.h@              for_ciris/psoc.c@
for_ciris/data_mgr.c@            for_ciris/psoc.h@
for_ciris/data_mgr.h@            for_ciris/rpc_tcp.h@
for_ciris/define.h@              for_ciris/rpc_xdr.c@
for_ciris/extern.h@              for_ciris/tcp_psoc.c
for_ciris/global.h@              for_ciris/tcp_tansaku.h@
```

これら処理モジュールを起動する LISP 関数は

```
c-interface2.lisp                    out_line_graph.lisp
ceps-access.lisp                     preprocess.lisp
ceps_ai_graph.lisp                   scale-space-filtering-data-io.lisp
ceps_edge_top_level.lisp             test.lisp
edge-detections.lisp
```

である．この中で重要なのは以下の 2 つのファイルである．

```
;;; 輪郭線解析のプログラム
preprocess.lisp
;;; 輪郭線解析結果 CEPS をアクセスするためのインタフェース関数
ceps-access.lisp
```

である．他に edge-detections.lisp がある．これはエッジ画像の任意領域からエッジを抽出するものである． 現在は使用していない．しかし，トップダウン解析には必要なモジュールである．

5.1 チェインコード生成モジュール

チェインコード生成のインタフェース関数は preprocess.lisp の中にある．チェインコード生成の最上位処理として，

```
;;; 左画像についてのチェインコード生成とスケールスペース処理
(defun left-image-preprocess nil
  (let*  ((tmp (edge-detecting *left-area-image-file* *left-edge-image-file*)))
    (setq *left-contour-root* (stack-all-lines (first tmp)))
```

```
    (setq *left-edge-root* (stack-all-lines (second tmp)))
    (call-ait-process *left-contour-root*)
    (call-ait-process *left-edge-root*)
    (free-lines-area (first tmp))
    (free-lines-area (second tmp))))

;;; 右画像についてのチェインコード生成とスケールスペース処理
(defun right-image-preprocess nil
  (let*  ((tmp (edge-detecting *right-area-image-file*
                               *right-edge-image-file*)))
    (setq *right-contour-root* (stack-all-lines (first tmp)))
    (setq *right-edge-root* (stack-all-lines (second tmp)))
    (call-ait-process *right-contour-root*)
    (call-ait-process *right-edge-root*)
    (free-lines-area (first tmp))
    (free-lines-area (second tmp))))
```

この処理では始めに手続き edge-detecting によりチェインコード生成を行なっている．この edge-detecting はファイルから領域画像，エッジ画像を呼んでいるが， 前処理結果を返すためのデータ構造 PRE_FILTER_DATA を p-area-im, p-edge-im の代りに用いれば良い． contour-detection は領域画像のポインターを受け取って，領域外周（領域 ID が 0 または 255 以外）をチェインコードとして返す． edge-detection はエッジのチェインコードを返す． その前に 一度得られた領域外周を再びエッジによって重複しないように contour-line-masking によってマスク処理をしている．

```
(defun  edge-detecting (area-file edge-file)
  (let* ((p-area-im (input-image-data area-file))
         (p-edge-im (input-image-data edge-file))
         (p-contours (contour-detection p-area-im 100))
         (dummy (contour-line-masking p-contours 255 p-area-im 2))
         (p-edges (edge-detection p-edge-im p-area-im 30 100 40 3)))
    (draw-all-lines "/tmp/test_out.im" p-contours p-edges)
;;    (image-file-out  "/tmp/test_out2.im" p-area-im)
    (free p-edge-im)
    (free p-area-im)
    (list p-contours p-edges)
))

;; 輪郭線の抽出
(define-c-function (contour-detection "_contour_detection")
  ((im :pointer)
```

```
   line-length-limit  ;;; この長さ以下の直線は検出しない 20
   ) :result-type :pointer)

;; エッジの抽出
(define-c-function (edge-detection "_edge_detection")
  ((im-edge :pointer)
   (im-area :pointer);; 領域画像, 0xff は不検出領域として予約されている.
   line-length-limit ;;; この長さ以下の直線は検出しない 10
   find-level        ;;; コントラストの閾値 この値以下の点は開始点としない 60
   trace-level       ;;; このコントラスト値以下の点は探索しない. 20
   minimum_length    ;;; 線検出の基礎となる直線の長さ 3
   ) :result-type :pointer)
```

5.2 スケールスペース解析

得られたチェインコードの集合は C の構造体で表現されている. これを stack-all-lines によって LISP 上では line-root なる構造で一つの画素系列を表現する.

```
(defstruct (line-root
            (:print-function
             (lambda (n st ignore)
               (format st
                       "<Pointer:~D, type:~A, hue:~D, legth:~D>~%"
                       (line-root-pointer n) (line-root-type n)
                       (line-root-hue n) (line-root-length n)))))
  (pointer nil)
  (type 'contour) ;; デフォルトで輪郭線
  (hue 0)
  (id 0)
  (length 0)
  (trnx nil))
```

この構造の最後の trnx はスケールスペース解析した結果を保持するポインタである. これは call-ait-process を呼ぶことによって処理され結果が格納される.

```
(defun call-ait-process (lines-data &optional (filename "ait"))
  (dolist (line lines-data)
          (when (line-root-p line)
                (let* ((trnx (init-pro filename (line-root-pointer line)))
                       (lower-sigma 5.0)
                       (upper-sigma (max
```

```
                                lower-sigma
                                (round
                                 (if (> (line-root-length line) 320.0)
                                     40.0
                                     (/ (line-root-length line) 8)))))
                        (result
                         (scale-space-filtering
                          trnx (coerce lower-sigma 'single-float)
                          (coerce upper-sigma 'single-float)
                          *dammy-threshold* *log-sw* *rpc-sw*
                          "ciris2,ciris3,")))
                    (phase1 trnx)
                    (it-data-make trnx)
                    (setf (line-root-trnx line) trnx))))
  (setf lines-data (sort lines-data #'> :key #'line-root-length)))

;;; スケールスペースフィルタリングを行なう
(define-c-function (scale-space-filtering "_tansaku")
  ((trnx :pointer) lower-sigma upper-sigma
   dammy log-sw rpc-sw remote-host-names-strings)
  :result-type :integer)
```

このようにスケールスペース処理はscale-space-filteringにより実行される．引数は 結果格納先ポインタ， スケールファクタσの下限値，上限値，ダミー（未使用），結果出力のスイッチ（通常は0），リモートプロシージャのスイッチ（0でローカル処理，1でリモート処理），リモートマシンの指定 である．スケールスペース処理は多数の浮動小数点演算が必要であるために 高速の計算機サーバーとして SGI の IRIS ワークステーションを使用している．

次にスケールスペース解析の結果をアクセスする関数を説明する．これらの関数群はceps-access.lisp の中にある．

```
;;;CEPS トップレベル区間の探索を行なう　仕様その 4　領域輪郭
(define-c-function (ceps-nodes-top-level "_ceps_nodes_top_level")
  (trnx max-number int-array) :result-type :integer)

;;; トップレベル区間の始点終点情報を返す為の LISP 関数 エッジ
(defun ceps-edge-top-level-c (trnx)
  (let* ((int-array (make-array 4 :element-type 'fixnum))
         (result (ceps-edge-top-level trnx int-array)))
    (unless (= result -1)
            (list (list (aref int-array 0)
                        (aref int-array 1))
```

```
                (list (aref int-array 2)
                      (aref int-array 3))))))

;;; スケールスペース区間データ
(defstruct (interval (:type list))
  (node-first nil)
  (node-second nil)
  (contour-interval '(0 0))
  (curvature 0)
  (line-regularity 0)
  (curve-regularity 0)
  (line nil))

;;; スケールスペース区間データを上の構造体として返す.
(defun interval-attribute-c (trnx start end)
                -------)

;;; 下位区間の探索を行なう（仕様その３）の為のLISP関数
;;; リターン値は
(defun devide-interval-c (trnx start end)
-------)
```

画素系列毎にスケールスペースに展開されたデータ構造を上の関数でアクセスする． これは最上位の区間を求めたのち再帰的に再分割することによって行なわれる．

6 High-level subsystem

本章ではASDSの画像の基本特徴の生成，構造化，モデルとの照合部分のモジュールを説明する．処理の大部分はルールで書かれている． 概略は次節で示し．特別に節目を要すると思われる，線分円弧データの仮定，ルール実行の優先度，モデルとの照合を後に説明する．

6.1 概要

全ての処理は$src/load.lispに書かれてある．この内容を説明することによって 処理内容を明らかにする． なお，領域画像，エッジ画像はすでにファイルとして求まっているとして話を進める．以下はload.lispの内容である．

```
;;;-*-Mode: LISP; Base: 10.; Package: ART-USER -*-
;(unless (find :X-window *features*)
;    (setf *features* (cons :X-window *features*)))
;;出力表示がX-windowであれば以上のコメントをアウトにする.
;;SUN Common-LISPのシステム用グローバル*features*により制御する.
```

```
;; なお出力先を SGI の DGL を用いる場合は X-window の feature 設定により.
;;X ではなく DGL が起動される.

(defvar *left-area-image* nil)        ;; 領域画像のポインタ
(defvar *left-shade-image* nil)       ;; 輝度画像のポインタ
(defvar *left-contour-root* nil)      ;; 領域輪郭の画素系列管理テーブル
(defvar *left-edge-root* nil)       ;; エッジの画素系列管理テーブル
(defvar *left-area-image-file* nil)  ;; 領域画像のファイル指定
(defvar *left-edge-image-file* nil)  ;; エッジ画像のファイル指定
(defvar *left-shade-image-file* nil) ;; 輝度画像のファイル指定

(defvar *right-area-image* nil)
(defvar *right-shade-image* nil)
(defvar *right-contour-root* nil)
(defvar *right-edge-root* nil)
(defvar *right-area-image-file* nil)
(defvar *right-edge-image-file* nil)
(defvar *right-shade-image-file* nil)

(setq *left-area-image-file* "/usr1/etoh/image/cs15-data/hito_l.area")
(setq *left-edge-image-file* "/usr1/etoh/image/cs15-data/hito_l.lg")
(setq *left-shade-image-file* "/usr1/etoh/image/cs15-data/hito_l.y")
(setq *right-area-image-file* "/usr1/etoh/image/cs15-data/hito_r.area")
(setq *right-edge-image-file*  "/usr1/etoh/image/cs15-data/hito_r.lg")
(setq *right-shade-image-file* "/usr1/etoh/image/cs15-data/hito_r.y")

(setq  *CAMERA-R* 9.5)    ;; カメラの焦点距離 単位 mm
(setq *CAMERA-LENS-D* 200.0)    ;; カメラレンズ間距離  単位 mm
(setq *Z-RANGE-MAX* -1000.0)    ;; 認識対象空間の最小奥行き
(setq *Z-RANGE-MIN*  -5000.0)  ;; 認識対象空間の最大奥行き

(setq *upper-sigma* 80.0)  ;; スケールスペース σ の最大値
(setq *lower-sigma* 3.0)  ;; スケールスペース σ の最小値

(load "/usr1/etoh/artwork/proj2/load-lisp")
;;; これにより, ルールから起動される LISP 関数がロードされる.
(load "/usr1/etoh/artwork/proj2/load-bottomup-rule")
;; ボトムアップの構造化処理がロードされる.
(load "/usr1/etoh/artwork/proj2/load-match-rule")
```

```
;; モデルとの照合処理がロードされる.

(kill-servers "ciris3" "ciris2")
;;;RPC で用いるサーバーを一度 Kill する.
(check-server-status "ciris3" "ciris2")
;;; サーバープログラムを再び走らせる.
(left-image-preprocess)
(right-image-preprocess)
;; これで左右画像の中レベル処理が起動される.

(setq *left-area-image* (input-image-data *left-area-image-file*))
(setq *right-area-image* (input-image-data *right-area-image-file*))
(setq *left-shade-image* (input-image-data *left-shade-image-file*))
(setq *right-shade-image* (input-image-data *right-shade-image-file*))
;;; 予め保存されていた, 画像をロードする.
;;prefiltering のモジュールより最初から低レベル処理を起動する時は
;; リターン値のポインタをそのまま使用する.

#+X-window (load "/usr1/etoh/artwork/proj2/X/Xfunc.lisp")
;;; これで X の関数が使えるようにスタートする.
;;;X-window の出力を行なう関数群をロードするプログラム
;;;DGL を使用するバージョンでは X-window の代りに DGL のプログラムをロード
;;; する.
#+X-window (Xfunc-start "cs15:0")
;;; ここでは cs15 の X サーバを起動している.
;;DGL バージョンでは例えば, (Xfunc-start "ciris7") とする.

(reset)
;;;ART のリセット
(time (run))
;; ルールベースの起動を行なう.

(hypothesize-new-ribbon-if-any 'ribbon537 -20 100)
(hypothesize-new-ribbon-if-any 'ribbon549 -20 100)
;;; 未検出リボンの探索,
;; 未検出リボンの探索はこのように現在はマニュアルで行なっている.
;; モデル照合ルーティンに組み込んで自動化する必要がある.
(run)
;; ルールベースの再起動
```

```
;;; 未検出リボンがあらたに発見された場合，アジェンダに発火待ちのルールが
;;; 溜っている．　再起動により，アジェンダ内をクリアする．
(search-interpretation-using-scheme)
;; モデルの照合状態を ART のスキーマ上で統合する．
(report-all-interpretations "result0604")
;;; 認識結果を result0604 のディレクトリ以下に書き込む
(write-out-result "result0604/extension.log")
;;; 認識結果のスコアを result0604/extension.log に書き込む
```

以上の load.lisp により，入力画像の設定，各種パラメータ等の設定を行なっている．処理内容の理解には load.lisp から呼ばれる load-lisp.lisp,load-bottomup-rule.lisp 及び，load-match-rule.lisp を参照すると良い．load-lisp.lisp の内容を次に示す．

```
;;;-*-Mode: LISP; Base: 10.; Package: ART-USER -*-
;;; 輪郭線データの構造記述や入力などの処理モジュール
(load "/usr1/etoh/artwork/proj2/input")
;;; 汎用 lisp モジュール
(load "/usr1/etoh/artwork/proj2/primitive")
;;; デンプスターシェイファーの確率算用モジュール
(load "/usr1/etoh/artwork/proj2/dempster")
;;; 画像データのアクセスに使用している c-program
;;; とのインタフェースモジュール
(load "/usr1/etoh/artwork/proj2/c-interface")

;;atms に関するモジュール
#+golden (load "/usr1/etoh/artwork/proj2/golden-atms")
#+golden (load "/usr1/etoh/artwork/proj2/golden-atms-art")
#-golden (load "/usr1/etoh/artwork/proj2/atms")
#-golden (load "/usr1/etoh/artwork/proj2/atms-art")

;;jtms に関するモジュール
(load "/usr1/etoh/artwork/proj2/jtms")
(load "/usr1/etoh/artwork/proj2/jtms-art")
(load "/usr1/etoh/artwork/proj2/jtms-handler")

;;; 2 次元データの幾何計算プログラム
(load "/usr1/etoh/artwork/proj2/geo")
;;;ribbon, cylinder の生成用の評価プログラム
(load "/usr1/etoh/artwork/proj2/geo2")
;;;cylinder とモデルの照合評価プログラム
```

```
(load "/usr1/etoh/artwork/proj2/geo3")
;;; 3次元幾何のプログラム
(load "/usr1/etoh/artwork/proj2/geo3d")
;;; 共線化のための幾何プログラム
(load "/usr1/etoh/artwork/proj2/collinear-lisp")
;;;cylinder の2次元投影プログラム
(load "/usr1/etoh/artwork/proj2/projection.lisp")

;;; 輪郭線解析のプログラム
(load "/usr1/etoh/artwork/preprocess-parallel/preprocess")
;;
;;; 輪郭線解析結果 CEPS をアクセスするためのインタフェース関数
(load "/usr1/etoh/artwork/preprocess-parallel/ceps-access")

;;;AIT 生成のためのユーティリティ関数群
(load "/usr1/etoh/artwork/proj2/ait-process")

;;; グラフ出力用のユーティリティ
(load "/usr1/etoh/artwork/preprocess-parallel/ceps_ai_graph")
(load "/usr1/etoh/artwork/preprocess-parallel/out_line_graph")

;;;foa を解析するルーティン
(load "/usr1/etoh/artwork/proj2/foa-analyse")

;;; 輪郭線や得られた軸線を出力するプログラム
(load "/usr1/etoh/artwork/proj2/output-ribbon-contours.lisp")
(load "/usr1/etoh/artwork/proj2/output-cylinders.lisp")

;;;; 最優先探索のためのモジュール
(load "/usr1/etoh/artwork/proj2/best-first-search")

;;;; 検出されないリボンを探索するためのモジュール
(load "/usr1/etoh/artwork/proj2/ribbon-hypothesizing")

(setq *contradiction-handler* (function handler-ex))
(setq *debug-jtms* nil)
(setq *debug-jtre* nil)

;;;; デバック用に log ファイルを常に append していることに注意！！！
```

```
(format t "CAUTION! log file is created and is being appended!~%")

p;;;atms,jtms, シルエットデータのロード等の初期化ルーティン
;;; この初期ルーティンは最後にくる必要がある.
(load "/usr1/etoh/artwork/proj2/init-data")
```

load-lisp.lisp 内のほとんどの関数は load-bottoomup-rule.lisp によってロードされるルールから起動される. load-bottoomup-rule.lisp の内容を次に示す.

```
;;;-*-Mode: LISP; Base: 10.; Package: ART-USER -*-
;;; スキーマの定義をロードする
(load "/usr1/etoh/artwork/proj2/load-schema")
;;; スケジューラの本体
(art-load "/usr1/etoh/artwork/proj2/scheduler.art")
;;;AIT 生成のルール集合
(art-load "/usr1/etoh/artwork/proj2/ait-gen.art")
;;;ribbon 生成の本体
(art-load "/usr1/etoh/artwork/proj2/parallel.art")
;;; 共線化の本体
(art-load "/usr1/etoh/artwork/proj2/collinear.art")
;;;cylnder 生成の本体
(art-load "/usr1/etoh/artwork/proj2/stereo.art")
;;; 閉領域仮説生成
(art-load "/usr1/etoh/artwork/proj2/closure.art")
;;; 閉領域のステレオマッチング仮説生成
(art-load "/usr1/etoh/artwork/proj2/stereo-match-plane.art")
;;;ribbon,cylinder の連結性のチェック
(art-load "/usr1/etoh/artwork/proj2/connect.art")
```

ルールの右手部における手続きは次のように定義されている.

- ;;; 帰結のスキーマ構成とその前提のスキーマ名のリストを入力する.
  ```
  (defun justify! (consequent antecedents &optional (source 'USER))
  ------)
  ```
- `(defun assume! (fact)  ---)`
- `(defun assert! (fact)  ---)`
- ;;nogood の fact のスキーマ名のリストだけを宣言する
  `(defun nogood! (facts) ---)`

ルールの実際例として 1 組の線分データから parallel(ism) と symmetry の空間関係を生成するルールを次に例示する.

```
(defrule check-parallelism
  "直線の平行仮説を生成する"
  (declare (salience *parallel-constraction-salience*))
  (schema PARALLELISM-GATE
          (ribbon-length-threshold ?length-limit))
        ;;組合せ数を減らすための閾値処理（ゲート）
          ;;スケジューラが変更する.
  (schema ?line0
          (instance-of line-data) ;;ラインデータのインスタンスとマッチ
          (type contour)
          (status IN) ;;;信念状態が ''信じられている'' ものとマッチ
          (hue ?hue)
          (belief ?belief0&:((first (list$ ?belief0)) > 0.70))
                ;;;D-S の belief 値を用いた枝刈り.
          (prior 0|1|2)
          (atms-node ?node0)  ;;atms-node を束縛
          (plane ?side)
          (length ?length0&:(> ?length0 ?length-limit))
                ;;長さによる枝刈り
          (head ?head0)
          (tail ?tail0)
          (orientation ?orientation0))
  (schema ?line1&:(string-greaterp ?line1 ?line0)
          (instance-of line-data)
          (type contour)
          (status IN)
          (belief ?belief1&:((first (list$ ?belief1)) > 0.70))
          (prior 0|1|2)
          (hue ?hue)
          (plane ?side)
          (length ?length1&:(> ?length1 ?length-limit))
          (atms-node ?node1&:(check-labels-by-nodes (list ?node0 ?node1)))
        ;;;;ここに注意！！ 上で束縛された atms-node を元に
        ;;;ラインデータの組合せが矛盾であるかどうかをチェックしている.
          (head ?head1)
          (tail ?tail1)
          (orientation ?orientation1))
  (not (schema ?
               (instance-of SYMMETRY)
```

```
            (lines ?line0)
            (lines ?line1)))
(not (schema ?
            (instance-of PARALLEL)
            (lines ?line0)
            (lines ?line1)))
=>
(let* ((line0 (makeline (list$ ?head0) (list$ ?tail0)
                        ?length0 (list$ ?orientation0)))
       (line1 (makeline (list$ ?head1) (list$ ?tail1)
                        ?length1 (list$ ?orientation1)))
       (result (make-ribbon-with-parallelism2 line0 line1 ?side ?hue)))
  (if result
    then
    (let* ((axis (nth 0 result))
           (base0 (nth 1 result))
           (base1 (nth 2 result))
           (area (nth 5 result))
           (sym-axis (nth 6 result))
           (contours (subseq result 1 5))
           (symmetry (gentemp "SYMMETRY")) ;; これで関係の具体名を生成
           (parallel (gentemp "PARALLEL")) ;; これで関係の具体名を生成
           (side0 (third contours))
           (side1 (fourth contours))
           (shading (check-distribution-ratio-c
                     (which-side? ?side *left-shade-image*
                                  *right-shade-image*)
                     ;;c-interface.lisp の関数コール
                     (/ (+ (line-sx side0) (line-ex side0)) 2)
                     (/ (+ (line-sy side0) (line-ey side0)) 2)
                     (/ (+ (line-sx side1) (line-ex side1)) 2)
                     (/ (+ (line-sy side1) (line-ey side1)) 2)))
      ;;輝度画像の分散を観察して parallel(ism) の関係を宣言するか
      ;;吟味している.
           (belief-symmetry
            (comb-belief (list (symmetry? line0 line1 sym-axis base0 base1)
                               (cond ((or (> shading 80) (< shading 2))
                                      '(0 1))
                                     (t '(0 0))))))))
```

```
      (belief-parallel
       (if (> shading 20) then '(0 1)
           else (parallel? line0 line1 sym-axis base0 base1))))
;; 以上により空間関係の信頼度を決定している.
  (assume! `(,symmetry (instance-of SYMMETRY)
                       (lines ,?line0 ,?line1)
                       (plane ,?side)
                       (belief ,(seq$ belief-symmetry))
                       (hue ,?hue)
                       (area ,area)
                       (axis ,(seq$ axis))
                       (shade ,shading)
                       (contours ,(seq$ contours))))
  (when (> (second belief-symmetry) 0.9)
    (nogood! `(,symmetry)))
;;D-S の disbelief が 0.9 以上であれば信念の翻意がおきる.
  (when (and (> (first (list$ ?belief0)) 0.9)
             (> (first (list$ ?belief1)) 0.9))
    (assume! `(,parallel (instance-of PARALLEL)
                         (lines ,?line0 ,?line1)
                         (plane ,?side)
                         (belief ,(seq$ belief-parallel))
                         (hue ,?hue)
                         (area ,area)
                         (axis ,(seq$ axis))
                         (shade ,shading)
                         (contours ,(seq$ contours))))
    (if (> (second belief-parallel) 0.9) then
        (nogood! `(,parallel)) else
        (when (not (> (second belief-symmetry) 0.9))
          (nogood! `(,symmetry ,parallel)))))))))
;;; 同時に成立しない関係を規定
;;; 本来は別ルールで表現するものであるが, 効率の点から
;;; 同じルール内に書いている.
```

以上処理のロードモジュールとルールの具体例を示すことにより, 高レベルの処理の概要を示した.

6.2 線分と円弧の仮定

線分と円弧はと $src/ait-process.lisp に書かれてある手続きにより，区間データを階層化する． この階層データに対して ait-gen.art に書かれてあるルールが自動発火することにより線分，円弧が仮定される．ここで処理パラメータとして

1. 区間の再帰的分割の基準
2. 直線度，円弧度の区間を各々線分，円弧としてデータベース内に仮定基準
3. 効率化のため，不要な線分，円弧の除外基準

がある． 1 に関して

```
(defvar *regularity-threshold* 0.025)  ;;;in ait-process.lisp
```

により，再帰的分割の深さを決定している． 2,3 に関しては， ait-gen.art 内のルールを参考にすると良い．例えば以下のルールではデータベース内に宣言されている 制御用データ

```
(schema LINE-REGULARITY ),  (schema CURVE-REGULARITY)
```

により宣言される線分，円弧を制御している．また グローバル変数 *max-arc-R* により， 径の大きな円弧の仮定を抑制している．

```
(defrule line-of-interval
  "ラインデータの宣言"
  (declare (salience (- *line-extraction-salience* 10)))
  (schema LINE-REGULARITY
          (instance-of schedule)
          (threshold ?thres0)
          (status IN))
  (schema CURVE-REGULARITY
          (instance-of schedule)
          (threshold ?thres1)
          (status IN))
  (schema ?interval
          (instance-of interval-data)
          (line-regularity ?line-degree)
          (curve-regularity ?curve-degree)
          (line ?line)
          (arc ?arc)
          (side ?side)
          (type ?type)
          (pixels ?pixels)
          (hue ?hue))
```

```
=>
(let ((line nil)
      (arc nil)
      (line-data (list$ ?line))
      (arc-data (if (eq ?arc 'NULL) then ?arc else (list$ ?arc))))
  (if (and (< ?line-degree ?thres0) (not (edge-line? line-data))) then
      (setq line (assume!-line line-data ?interval
                               ?side ?type ?hue ?line-degree ?pixels)))
  (if (and (not (eq ?arc 'NULL)) (arc-R arc-data)  (< ?curve-degree ?thres1)
           (< (arc-R arc-data) *max-arc-R*)
           (> (arc-R arc-data) *min-arc-R*)
           (not (schemap (intern (concatenate 'string
                                   "ARC" (subseq (string ?interval) 8))))))
    then
    (setq arc (assume!-arc arc-data ?interval ?line
                           ?side ?type ?hue ?curve-degree ?pixels)))
  (if (and line arc) then
      (nogood! `(,line ,arc)))))
```

6.3 ルールの優先度

ARTでは"salience"と呼ばれるアジェンダ内でルールの実行優先度を制御する値を行なっている（ルール左手部先頭の(delcare...)に注意）．これは$src/init-data.lispに以下のように示してある． ARTでは動的な優先度変更はできない． この点がシステム構築の上では融通性が低い． ASDSでは 矛盾発見のルール実行優先度をもっとも高く，スケジューラに関するルール実行の優先度を低くすることによって システムを構成している．

```
(defvar *jtms-nogood-salience* (- *maximum-salience* 0))
(defvar *atms-nogood-salience* (- *maximum-salience* 1000))
(defvar *scheduler-control-salience* (- *maximum-salience* 2000))
(defvar *bottom-up-control-salience* (- *maximum-salience* 3000))
(defvar *line-extraction-salience* (+ *default-salience* 2000))
(defvar *parallel-constraction-salience* (+ *default-salience* 0))
(defvar *collinear-constraction-salience* (+ *default-salience* 1000))
(defvar *stereo-match-salience* (- *default-salience* 500))
(defvar *connection-constraction-salience* (- *default-salience* 1000))
(defvar *bottom-up-check-salience* (- *default-salience* 1200))
(defvar *model-match-control-salience* (- *default-salience* 1500))
(defvar *model-match-salience-0* (- *default-salience* 2000))
(defvar *model-match-salience-1* (- *default-salience* 2001))
(defvar *model-match-salience-2* (- *default-salience* 2002))
```

```
(defvar *model-match-check-salience* (- *default-salience* 3000))
(defvar *model-match-check-salience-1* (- *default-salience* 3010))
(defvar *scheduler-check-salience* (+ *minimum-salience* 5))
(defvar *scheduler-last-salience* (+ *minimum-salience* 0))
```

6.4 モデルとの照合

モデルとの照合は $src/load-match-rule.lisp によって interpretation.art がロードされる． ここに書かれている記述はマネキン人形の認識のためのものである． interpretation.art 内の記述を以下に説明する．

```
(assert! '(HEAD
   (is-a PART)))
;;;---------- 省略 --------------------
(assert! '(CALF
   (is-a PART)))

(assert! '(FOOT
   (is-a PART)))

(assert! '(CRT
   (is-a PART)))
```

このように モデルデータが 事実として データベース内に宣言されている．モデルとの照合方法として モデルデータと 3 次元のシーン記述要素とをインタプリター方式で 逐次照合しながら 行なうことができるが，インプリメントの利便性から，ルールの中に 照合と制約を埋め込んだ．以下は 腕部との照合を仮定するルールである．実際はシステム効率の観点から全ての部分を一度に探索するようにルールを記述している．

```
(defrule human-body-detector
  "腕を見つける "
  (declare (salience *model-match-salience-0*))
  (schema ?cylinder
          (instance-of CYLINDER)
          (status IN)
          (belief ?belief0&:(> (nth$ ?belief0 1) 0.5))
          (axis ?axis)
          (L-diameter ?Ld)
          (R-diameter ?Rd)
          (hue ?hue&:(and (> ?hue 0) (< ?hue 50)))
          (pixels ?pixels))
  =>
  (let* ((axis (list*$ ?axis))
```

```
        (Ld (list$ ?Ld))
        (Rd (list$ ?Rd))
        (Bd (/ (+ (first Ld) (first Rd)) 2))
        (Td (/ (+ (second Ld) (second Rd)) 2))
        (maxd (max Bd Td))
        (mind (min Bd Td))
        (length (line-3d-length axis))
        (head-y (second (line-3d-head axis)))
        (tail-y (second (line-3d-tail axis)))
        (direction (line-3d-direction axis))
        (match-list nil))
;;;----------省略----------
    (when (and (< length 300)
               (< head-y (- *ground-level* 300))
               (< tail-y (- *ground-level* 300))
               (> head-y *head-level* )
               (> tail-y *head-level* )
               (< maxd 100)
               (> mind 20))
;; ここでは同時に前腕，上腕の対応を仮定している.
;; 実際のインプリメントでは仮説生成ルールと導出ルールを分けずに
;; 効率の観点から 両者は同じルール上で実現しいてる.
      (let ((label-name1 (gentemp "FOREARM"))
            (match-name1 (gentemp "MATCH"))
            (label-name2 (gentemp "UPPER-ARM"))
            (match-name2 (gentemp "MATCH")))
        (push (assume! `( ,match-name1
                          (instance-of MODEL-MATCH)
                          (3D-OBJECT ,?cylinder)
                          (MODEL-INSTANCE FOREARM)))  match-list)
        (push (assume! `( ,match-name2
                          (instance-of MODEL-MATCH)
                          (3D-OBJECT ,?cylinder)
                          (MODEL-INSTANCE UPPER-ARM))) match-list)
        (justify! `(,label-name1
                    (instance-of LABEL)
                    (3D-OBJECT ,?cylinder)
                    (match-assumption ,match-name1)
                    (bwd ,?cylinder)
```

```
                    (bwd FOREARM)
                    (PIXELS ,?pixels))
                  (list ?cylinder match-name1))
            (referent-schema-value ?cylinder 'fwd label-name1)
        (referent-schema-value 'FOREARM 'fwd label-name1)
        (justify! '(,label-name2
                    (instance-of LABEL)
                    (3D-OBJECT ,?cylinder)
                    (match-assumption ,match-name2)
                    (bwd ,?cylinder)
                    (bwd UPPER-ARM)
                    (PIXELS ,?pixels))
                  (list ?cylinder match-name2))
        (referent-schema-value ?cylinder 'fwd label-name2)
        (referent-schema-value 'UPPER-ARM 'fwd label-name2)))
;;;;;;---------------- 省略  ------------------------
    (nogood-binary-relation match-list)))
;;; 同じシリンダに複数の解釈は同時に存在しないため Nogood としている.
```

次に 両立しない モデル部品と記述要素との対応を宣言する制約ルールの例を以下に示す. このルールでは 上腕と胴体間に接続関係（ボトムアップルールにより，画像上，空間上で宣言される）がない場合は そのような対応関係は両立しないことを宣言している.

```
(defrule upper-arm-body-constraints
  "関係によるチェック"
  (declare (salience *model-match-salience-0*))
  (schema ?arm
          (instance-of MODEL-MATCH)
          (status IN)
          (model-instance UPPER-ARM)
          (atms-node ?nodeA)
          (3D-object ?cylinderA))
  (schema ?trunk
          (instance-of MODEL-MATCH)
          (status IN)
          (model-instance TRUNK)
          (atms-node ?nodeB)
          (3D-object ?cylinderB)
          (atms-node ?nodeB&:(check-labels-by-nodes  (list ?nodeA ?nodeB))))
  (not (schema ?
```

```
            (instance-of CYLINDER-CONNECTIVITY)
            (status IN)
            (cylinders ?cylinderA)
            (cylinders ?cylinderB)))
  =>
  (if (check-labels (list ?cylinderA ?cylinderB)) then
    (nogood! '(,?arm ,?trunk))))
```

以上のようにシーン解釈ではシーン記述のルールに加えて

1. モデルの各部分とシリンダ・平面とをその属性（軸長，見かけの太さ，辺長，色等）により対応を仮定する仮説生成ルール
2. 得られた対応と画像特徴を統合し新たなコンテクストを生成する導出ルール
3. 複数の対応の中から空間的配置により両立しない組を宣言する制約ルール

を記述することにより，極大コンテクストとして得ることができる．

参考文献

[1] 宮脇, 石橋, 岸野：“色彩情報を用いたカラー画像の領域分割”, 信学技報, IE89-50, pp.43-48, (Sep. 1989).

[2] 栄藤, 伴野, 小林: “ステレオ輪郭像を用いた円錐体モデルの再構成”, 信学技報, PRU89-36, pp.69-76, (1989).

[3] M.Etoh, A.Tomono & Y.Kobayashi: “Cylindrical part recognition in occluding contours”,*SPIE Proc. Intelligent Robots and Computer Vision VIII, Vol. 1192*, pp.353-362, (Nov. 1989).

[4] 栄藤, 伴野, 岸野: “ステレオ輪郭像を入力とした物体の一般化円筒複合体による記述”, 信学論, Vol.J73-D-2, No.9 (掲載予定, 1990).

[5] 栄藤 稔, 岸野 文郎: “仮説に基づくシーン記述”, 信学技報, AI90-55, , pp.61-68, (July 1990).

[6] M.L.Ginsberg: “Non-monotonic Reasoning using Dempster's rule”, *Proc. National Conf. on Artificial Intelligence, Vol.1984*, pp.126-129, (1986).

[7] Y. Shirai: “Three-Dimensional Computer Vision”, pp.74-78, Springer-Verlag, (1987).

[8] H.Harashima, K.Aizawa & T.Saito: “Model-based synthesis coding of videotelephone images-conception and basic study of intelligent image coding” *Trans. IEICE of Japan, Vol. E-72*, No.5, (May 1989).

[9] S.W.Zuker, A.Rosenfeld & L.S. Davis: “General purpose models: expectation about the unexpected”, *Proc. 4th Int. Joint Conf. on Artificail Intell.*, pp.716-721, (1975).

[10] D.Lowe: “Three-Dimensional Object Recognition from Single Two-Dimensional Images” *Artificial Intelligence, Vol.31.* pp.355-395 (1987).

[11] H.S. Lim & T.O. Binford: "Stereo Correspondence: A hierachical approach", *Proc. DARPA Image Understanding Workshop*, pp.234-241, (1987).

[12] R. Mohan & R.Nevatia : "Using perceptual organization to Extract 3-D structures", *IEEE trans. on Pattern anal. & mach. intell., PAMI-11*, pp.1121-1139, (1989).

[13] M.Herman & T.Kanade: "Incremental reconstruction of 3-D scenes from multiple, complex images" , *Artificial Intelligence, Vol. 30*, pp.289-341, (1986).

[14] 伊庭, 松原, 井上: "環境モデルにおける物体の見え方と見方", 人口知能学会誌, Vol.3, No.4, pp474-485, (July 1988).

[15] J.Doyle: "A Truth Maintenance System", *Artificial Intelligence, Vol. 12*, No.3 , pp.231-272, 1979.

[16] J. de Kleer: "An Assumption-based TMS" "Extending the ATMS" "Problem Solving with the ATMS", *Artificial Intelligence, Vol.28*, pp.127-224, 1986.

[17] J. de Kleer: "Back to Backtracking: Controling the ATMS", *Proc. Natl. Conf. Artif. Intell. Vol.1986*, no.2, 1986.

[18] G.M.Provan: "Model-Based Object Recognition:A Truth Maintenance Approach", *Proc. 4th Conf. on Artificial Intelligence Applications*, Mar.14-18, (1988).

[19] G.M.Provan: "Efficiency analysis of multiple-context TMSs in scene representation", *Proc. Natl. Conf. Artif. Intell. Vol.1987*, No.1, pp.173-177. (1987).

[20] K.D.Forbus: "Building Problem Solvers: Program notes on Truth Maintenance Systems", *AAAI-87 tutorial, TA-4*, pp.294-323, (1987).

[21] K.D.Forbus & J. de Kleer: "Forcusing the ATMS", *Proc. Natl. Conf. Artif. Intell. Vol.1988*,no.1, pp.193-198, (1988).

[22] D. Marr & E. Hildreth: "Theory of edge detection", *Proc. R. Soc. London, B. 207*, pp.187-217, (1980).

[23] R.M.Haralick & L.G.Sharpiro: "The consistent labeling problem:part 1" *IEEE, Trans. Pattern Anal. & Mach. Intell., Vol. PAMI-1*, No.2,pp.173-184 (1979)

Appendix: デモシステム

デモシステムは ciris7 のワークステーションより利用する．ここではボタンフライの機能を利用して マウスのクリック操作を基本として利用することができる．デモシステムの操作は以下のステップからなっている．

1. ciris7 にユーザー名 desi(Demonstration Environment for Scene Interpretation) でログインする．
2. “demo” と入力すればデモ用のボタンフライが起動される．
3. 添付（CSK 吉川氏作成）の資料を参考にして art のボタンをクリックする．これにより cs40 にリモートログインした状態のウィンドウが開かれる．
4. そのウィンドウにおいて “artdemo” とキーインする．これは 画像の初期処理，エッジ，領域外周をすでにスケールスペース展開した状態で “disksave” したものである．
5. LISP の入力待ちのプロンプトが表示された段階で “(do-run)” とキーインする．これにより認識処理がスタートする．

注意しなければならないことは，cs40 でのプログラムが異常終了した場合，セマフォーや共有メモリーが確保されたまま放置される危険性がある．コマンド “ipcs” で確かめた後，放置された変数，フラグがある場合はコマンド “ipcrm” より クリアしておく必要がある．

ｂｕｔｔｏｎｆｌｙの説明とメニューデータの構成

このデモシステムでは、実行時の各種選択に”ｂｕｔｔｏｎｆｌｙ”というメニュー選択のプログラムを使用している。

このプログラムはＳＧＩのＩＲＩＳに用意されていたもので、指定の形式でデータを用意することによってボタンの色や説明の内容や、実行する命令やプログラムを指定する事が出来る。

”ｂｕｔｔｏｎｆｌｙ”ではメニューを階層的に指定できるため、デモの為のメニューが大変分かり辛くなっています。そこで以下に子のでもデモで使用している各種メニューの説明と、それらのメニューの階層関係の説明をします。

”ｂｕｔｔｏｎｆｌｙ”の機能として、ポストスクリプトで書いたヘルプメッセージの指定も出来ます。説明用のデータを貯めるために（~desi/demo-main/data/infos）というディレクトリーを用意しましたが、現在は何も入ってません。

”ｂｕｔｔｏｎｆｌｙ”自体の説明はＩＲＩＳのマニュアルを参照して下さい。
(IRIS-4D User's Reference Manual Volume 2 Section 6 BUTTONFLY(6D))

1　デモ用のメニューの起動。

ｃｉｒｉｓ７にコンソールからｄｅｓｉでログインして、コマンド入力待に”ｄｅｍｏ”と入力して下さい。（ciris7:~desi/etc/demo）が動いてｂｕｔｔｏｎｆｌｙの”マウスによる位置決め”になります。

（ciris7:~desi/etc/demo）は、シェルスクリプトで、環境変数で設定されたコマンドサーチパスに従ってこれが動きます。

この時、ｂｕｔｔｏｎｆｌｙには、（~/demo-main/data/menus/menu_demo）という引数が与えられ、この引数で指定されたファイルが最初のボタンのデータになります。

2　ボタンデータの階層。

以下に（~desi/demo-main/data/menus/menu_demo）から始まるデモメニューの階層説明を書きます。
説明の都合上、現在のディレクトリーは（ciris7:~desi/demo-main/data/menus）とします。
以下の名前はボタンのラベルを表わし（）内にそのボタンについて書かれているファイル名をこのディレクトリーから指定形で書きます。

```
起 動
  |
  +----  Viewer (menu_demo)  画像データを表示選択。
  |       +----  RGB (m_viewer_1_rgb)  ＲＧＢ画像を表示選択。
  |       |       +----  RGB-l (m_rgb_l)  左データを表示する。
  |       |       +----  RGB-r (m_rgb_r)  右データを表示する。
  |       |
  |       +----  H (m_viewer_2_h)  色相画像を表示選択。
  |       |       +----  H-l (m_h_l)  左データを表示する。
  |       |       +----  H-r (m_h_r)  右データを表示する。
  |       |
  |       +----  Y (m_viewer_3_y)  輝度画像を表示選択。
  |       |       +----  Y-l (m_y_l)  左データを表示する。
  |       |       +----  Y-r (m_y_r)  右データを表示する。
  |       |
  |       +----  D (m_viewer_4_d)  色彩画像を表示選択。
  |       |       +----  D-l (m_d_l)  左データを表示する。
  |       |       +----  D-r (m_d_r)  右データを表示する。
  |       |
  |       +----  AREA (m_viewer_5_area)  領域画像を選択。
  |       |       +----  AREA-l (m_area_l)  左データを表示する。
  |       |       +----  AREA-r (m_area_r)  右データを表示する。
  |       |
  |       +----  LG (m_viewer_6_lg)  エッジ画像を選択。
  |       |       +----  LG-l (m_lg_l)  左データを表示する。
  |       |       +----  LG-r (m_lg_r)  右データを表示する。
  |       |
  |       +----  GRAPH (m_viewer_7_graph)  元データと、ＡＩＴの表示を選択。
  |               +----  CONTOUR (graph_menus/mg_cont)  輪郭データの表示を選択。
  |               |       +----  LEFT (graph_menus/mgc_l)  左データを選択。
  |               |       |       +---- leftcont3 (graph_menus/mgcl_0)左輪郭データ番号３を表示
  |               |       |       +---- leftcont1 (graph_menus/mgcl_1)左輪郭データ番号１を表示
  |               |       |       +---- leftcont2 (graph_menus/mgcl_2)左輪郭データ番号２を表示
  |               |       |       +---- leftcont0 (graph_menus/mgcl_3)左輪郭データ番号０を表示
  |               |       |
  |               |       +----  RIGHT (graph_menus/mgc_r)  右データを選択。
  |               |               +--- rightcont3 (graph_menus/mgcr_0)右輪郭データ番号３を表示
  |               |               +--- rightcont1 (graph_menus/mgcr_1)右輪郭データ番号１を表示
  |               |               +--- rightcont2 (graph_menus/mgcr_2)右輪郭データ番号２を表示
  |               |               +--- rightcont0 (graph_menus/mgcr_3)右輪郭データ番号０を表示
  |               |
  1               3
```

```
1                3
|                |
|                +-----  EDGE (graph_menus/mg_edge)  ｅｄｇｅデータの表示を選択。
|                        +----  LEFT (graph_menus/mge_l)  左データを選択。
|                        |       +---- leftedge0 (graph_menus/mgel_0)左線データ番号０を表示
|                        |       +---- leftedge3 (graph_menus/mgel_1)左線データ番号３を表示
|                        |       +---- leftedge4 (graph_menus/mgel_2)左線データ番号４を表示
|                        |       +---- leftedge1 (graph_menus/mgcl_3)左線データ番号１を表示
|                        |       +---- leftedge2 (graph_menus/mgel_2)左線データ番号２を表示
|                        |
|                        +----  RIGHT (graph_menus/mge_r)  右データを選択。
|                                +--- rightedge0 (graph_menus/mger_0)右線データ番号０を表示
|                                +--- rightedge7 (graph_menus/mger_1)右線データ番号７を表示
|                                +--- rightedge8 (graph_menus/mger_2)右線データ番号８を表示
|                                +--- rightedge1 (graph_menus/mger_3)右線データ番号１を表示
|                                +--- rightedge6 (graph_menus/mger_4)右線データ番号６を表示
|                                +--- rightedge2 (graph_menus/mger_5)右線データ番号２を表示
|                                +--- rightedge4 (graph_menus/mger_6)右線データ番号４を表示
|                                +--- rightedge3 (graph_menus/mger_7)右線データ番号３を表示
|                                +--- rightedge5 (graph_menus/mger_8)右線データ番号５を表示
|
+----  art (menu_demo)  ＡＲＴ操作選択。
        +----  art (m_art_1)  ｃｓ４０にリモートログインした新しいウィンドウをオープンする。
        +----  result (m_art_2)  結果の表示をする。
```